Zadaci za vežbanje – takmičenje sa evaluatorom

1. A Sky Full of Stars

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
2 s	1000 MB	standardni ulaz	standardni izlaz

Za prirodan broj kažemo da je lep ukoliko je u njemu rastojanje između svake dve iste cifre bar 10. Rastojanje između dve cifre jednako je broju cifara između njih + 1 (recimo, u broju 12342, rastojanje između cifara '2' je 3). Imamo početni broj od n cifara i želimo da od njega napravimo lep broj. U jednom potezu moguće je obrisati bilo koju cifru početnog broja i na njeno mesto upisati neku drugu cifru. Koliko je najmanje poteza potrebno da bi dobili lep broj?

Ulaz: U prvom redu ulaza se nalazi se jedan prirodan broj n koji predstavlja broj cifara početnog broja. U sledećem redu se nalazi početni broj. Između cifara ne postoji razmak i broj može imati vodeće nule.
Izlaz: U prvom i jedinom redu izlaza ispisati najmanji broj poteza potrebnih da se od datog broja dobije lep broj.

Ograničenja $1 \leq n \leq 1.000.000$

Ulaz

8

00346731

Izlaz

2

Napomena: Broj 00346731 ne moze biti lep jer nam smetaju redom grupe podcifara 00, 34673.
Od tog broja se moze napraviti vise lepih brojeva, ali u ne manje od 2 poteza.
Sa 2 poteza mogu se dobiti lepi brojevi 09346721, 02346791, 05346781,...

Rešenje;

Ideja 1:

Uočimo da niz cifara broja mora da bude 10-periodican da bi uslovi bili ispunjeni, tj. num[i] = num[i + 10].

Ideja je da pamtimo broj određenih cifara na svakom i mod 10 mestu, Zato na kraju tražimo najbolji moguci poredak, tj. najcešcu cifru na svakom i mod 10 mestu. onda oduzimamo od ukupnog broja i mod 10 cifara (n/10 + (n % 10 > i)) broj ponavljanja najcešce cifre. Tako dobijamo koliko cemo puta morati da promenimo cifru na i mod 10 mestu. Uradimo za svako 0 <= i < 10.

Mana ideje 1: ovaj metod ne daje najoptimalnije rešenje, jer može doci do sukobljavanja cifara za neku poziciju. (npr. na 5 mod 10 poziciji ima 20 cifara 3 i 20 cifara 2, ovaj algoritam slucajno odabere jedan od njih, a npr. sledece najviše pojavljivanje cifre 3 je 1, a cifre 2 19. mi izaberemo 2 -> 20, 3 -> 1, a optimalnije je 3 -> 20, 2 -> 19).
Kako popraviti ovu manu?
Bruteforce = 10!, svaka moguca kombinacija ~ 3 000 000 * 10 (zbog proveravanja niza).

Ideja 2:
Backtracking

Breaking The Habit

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	1000 MB	standardni ulaz	standardni izlaz

Bora je otet na jednoj raskrsnici u njemu nepoznatom gradu. Otmičaru su ga strpali u auto i voze ga na neko tajno mesto. Oni znaju da Bora ne poznaje taj grad, pa mu nisu zavezali oči, a Bora je odlučio da zapamti kojim putem ga voze na sledeći način: Grad se sastoji od pravougaone mreže oblakodera. Između svaka dva susedna oblakodera nalazi se ulica. Ulice su numerisane brojevima, počevši od broja 1, odozgo prema dole i sa leva na desno. Kada ga otmičari voze po nekoj ulici, Bora u tajnosti zapiše u mobilni telefon visinu oblakodera sa leve i visinu oblakodera sa desne strane ulice. Nakon što auto pređe preko neke raskrsnice, Bora ponovo zapamti visinu sa leve i visinu sa desne strane. Na svakoj raskrsnici auto može nastaviti pravo, skrenuti levo, skrenuti desno, ili se polukružno okrenuti (i nastaviti ulicom odakle je došao). Kada su ga otmičari doveli na cilj, Bora je uspeo da pošalje policiji poruku sa podacima o visinama koje je zapisao.

Napisati program koji će pomoći policiji da otkrije koordinate raskrsnice na koju su otmičari odveli Boru.

Ulaz

U prvom redu ulaza se nalaze dva cela broja R i K, koji predstavljaju broj redova i broj kolona gradske mreže.

U svakom od sledećih R redova se nalazi po K celih brojeva, visine oblakodera redom.

U sledećem redu nalazi se celi broj N, dužina puta kojim se vozio Bora.

U sledećem redu nalazi se N brojeva, visine oblakodera koje je Bora video sa leve strane.

U sledećem redu nalazi se N brojeva, visine oblakodera koje je Bora video sa desne strane.

Izlaz

U prvom i jedinom redu izlaza treba ispisati koordinate (prvo red pa kolonu) raskrsnice gde su otmičari odveli Boru.

Ako postoji više rešenja, ispisati bilo koje.

Ograničenja

$3 \le R \le 100$

$3 \le K \le 100$

$1 \le N \le 10{,}000$

Visina svakog oblakodera je veća ili jednaka od 1, a manja ili jednaka od 10,000.

Ulaz

```
4 4
7 3 5 4
2 8 9 7
3 5 2 8
1 3 5 7
5
5 9 8 2 9
2 2 2 8 7
```

Izlaz

```
1 3
```

Ideja 1:
Za svako stanje na putu BFS-om proverimo gde sve mozemo stici
(levo, pravo, desno, polukruzno). Dakle, to bi bilo N*R*K provera, ali kako je vrednost 10000*100*100 na relativni na granici zbog vremenskog ogranicenja, onda koristimo Queue u kojm pamtimo gde sve mozemo biti u odnosu na tekuci poziv i na taj nacin ne proveravamo sve lokacije.

Slozenost algoritma je O(broj_pomeraja*n*m). Najgori slučaj je ako su visine svih zgrada jednake.

Ideja 2:
BFS uz ograničenje da dozvoljavamo posećivanje čvorova ako smo ih ranije posetili (ne u trenutnoj iteraciji).
Na početku generišemo sve moguce čvorove do kojih smo mogli da stignemo sa prvom levom i desnom visinom, a onda nastavljamo od tih čvorova na one čvorove koje možemo. Kako u jednoj iteraciji nećemo posetiti nijedan čvor više od jednom, a postoji n iteracija, složenost je O(nrk).

Castle of Glass

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	64 MB	standardni ulaz	standardni izlaz

https://petljamedia.blob.core.windows.net/root/Media/Default/Problem/BCup5-Problem%20G%20Matrix.pdf

You are given a square binary matrix of dimension . Elements on the main diagonal are all ones. We want to compute the th power of this matrix (MDCS written in ASCII codes is). To make things more interesting, we will define binary operations (addition) and (multiplication) as the following: + 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 1 So basically, addition is logical OR and multiplication is logical AND. The input matrix is too big for normal time constraints, so it will be given by listing all positions of ones in it. Also, for the output only the number of ones in the th power of the matrix is sufficient.

Rešenje:

Najpre primetimo da naš problem nema direktne veze sa matricama tj. treba da nađemo tranzitivno zatvorenje usmerenog grafa u kojem su čvorovi brojevi od 1 do n, a grana između njih postoji ako i samo ako se na mestu reda (prvi čvor) i kolone (drugi čvor) nalazi jedinica.

Objašnjenje u postoji i u BubbleBee booklet-u.

Što se tiče tranzitivnog zatvorenja, najbolji algoritam je Purdomov algoritam,
čija je složenost O(m + μn), gde je μ broj grana koje povezuju jako povezane komponente usmerenog grafa G.

Ovaj algoritam je korišcen i objašnjen u Bubblecup bookletu. Medutim, za grafove za velikim brojem jako povezanih komponenti, koje su same kompletni ili jako gusti grafovi (npr. najgori slucaj sqrt(n) komponenti, n grana u svakoj komponenti, veliki broj grana izmedu samih komponenti), algoritam poprima složenost O(mn), a za tu složenost imamo mnogo jednostavniji algoritam.

U stvari, algoritam je prosto dfs. Medutim, u ovom slucaju ćemo morati da uradimo nekoliko optimizacija kako bi smanjili vreme izvršavanja nekoliko puta, jer je M ~ 200000 i n ~ 5000.

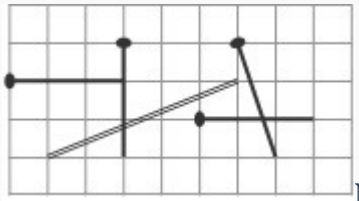
DFS za tranzitivno zatvorenje radi tako što uradi dfs n puta, svaki put pocevši iz različitog cvora. Ukoliko uspe da dođe do nekog čvora, on proveri da li je već uspeo da dođe do tog čvora u ovom koraku. Ako jeste, onda se vrati unazad. Ako nije, onda postavi da je uspeo da dođe to tog čvora od početnog čvora, i nastavi dfs dalje. Kako za svaki čvor obilazimo ceo graf, to daje složenost O(mn).

Međutim, možemo optimizovati algoritam i smanjiti vreme izvršavanja (samo u prosečnom slučaju, složenost i dalje ostaje ista u najgorem slučaju) ukoliko razmotrimo ovu situaciju.

Ako čvor broj k dođe do čvora l, l < k, on će moci da "preuzme" sve čvorove koje smo mogli da dostignemo iz cvora l. Međutim, i ako ovo dodatno optimizuje algoritam, on zahteva da izvršimo n koraka, što može da predstavlja problem ako ovaj proces budemo ponavljali više puta, ili na nezgodnom mestu (recimo, jedan cvor pre nego što bi dfs završio). Pošto smo konstruisali matricu tako da za svako i, matrix[i] predstavlja niz bitova koji na j-tom bitu čuva da li je čvor j dostižan iz i, vidimo da možemo da primenimo jedan trik: ako podelimo naš niz tako da svaki segment od 64 bita oznacimo sa unsigned long long, moći ćemo da koristimo ugrađene operacije bitske disjunkcije, i time smanjimo složenost "preuzimanja" 64 puta! Ovo svakako poboljšava vreme izvršavanja i sada će algoritam raditi mnogo efikasnije.

Despacito

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
0,5 s	512 MB	standardni ulaz	standardni izlaz

 Na datoj slici, dvostrukom linijom je predstavljen provodnik pod naponom. Provodnik se ne sme dodirnuti, kao i svaki metalni element izložen provodniku. Ali, oko provodnika su rasporedjene posebne metalne iglice.

Ulaz

U prvoj liniji standardnog ulaza dat je ukupan broj iglica i provodnika. U svakoj zasebnoj narednoj liniji standardnog ulaza, opisana je iglica preko dva para brojeva: x i y koordinata krajeva. Poslednja linija standardnog ulaza sadrži koordinate tačaka koje predstavljaju krajeve provodnika. Sve date koordinate su celi brojevi iz segmenta [0, 10000]. Ne postoji više od 1000 iglica.

Izlaz

U jedinoj liniji standardnog izlaza ispisati tačno jedan broj – broj iglica koje se mogu bezbedno dotaći golim rukama (u smislu da te iglice nemaju elektro veze sa provodnikom).

Ulaz

```
5

5 2 8 2
3 4 3 1

7 1 6 4

3 3 0 3

6 3 1 1
```

Izlaz

```
2
```

Rešenje:

Potrebno je naći sve iglice povezane sa provodnikom.

Zato ćemo praviti graf čiji će čvorovi biti iglice i provodnik, a grane između dva čvora pravimo ako se iglice/provodnik koje oni predstavljaju seku.

Da bi uspostavili graf, moracemo da obavimo n^2 koraka, što bi trebalo da bude dovoljno za ovo ograničenje. Moracemo da ispitamo presek svake dve iglice, tj. iglice i provodnika, i povežemo ih

ukoliko njihov presek nije prazan. Na kraju cemo u O(n) koraka ispitati presek provodnika sa iglicama jednim prolaskom dfs-a.

Presek dve iglice može predstavljati problem ukoliko su one paralelne, jer ce formula tada zahtevati da delimo sa 0, pa cemo morati da razmotrimo paralelnost pre traženja preseka.

```
#include <iostream>
using namespace std;

struct Point
{ int x, y; };

const int NMAX = 1024;

Point A[NMAX], B[NMAX];
int n;

bool adj[NMAX][NMAX];
bool visited[NMAX];

int compSize;

int direction(Point A, Point B, Point C)
{ int a1=B.x-A.x, a2=B.y-A.y;
  int b1=C.x-A.x, b2=C.y-A.y;
  int p=a1*b2, q=a2*b1;
  if(p>q) return +1;
  if(p<q) return -1;
  return 0;
}

bool onSegment(Point A,Point B,Point C)
{
  return min(A.x,B.x)<=C.x && C.x<=max(A.x,B.x) &&   min(A.y,B.y)<=C.y && C.y<=max(A.y,B.y);
}

bool intersect(Point A, Point B, Point C, Point D)
{ int d1 = direction(A,B,C);
  int d2 = direction(A,B,D);
  int d3 = direction(C,D,A);
  int d4 = direction(C,D,B);
  if (d1*d2<0 && d3*d4<0)return true;
  if (d1==0 && onSegment(A,B,C)) return true;
  if (d2==0 && onSegment(A,B,D)) return true;
  if (d3==0 && onSegment(C,D,A)) return true;
  if (d4==0 && onSegment(C,D,B)) return true;
  return false;
}

void dfs(int i)
{
   compSize++;
```

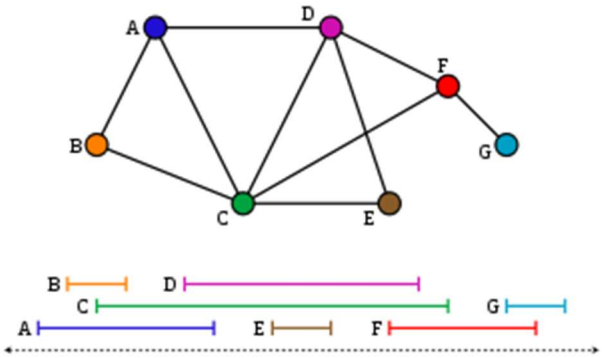
```
  visited[i]=true;
  for(int j=0; j<n; j++)
    if(adj[i][j] & !visited[j]) dfs(j);
}

int main()
{
   int i;
  cin >> n;
  for(i=0;i<n;i++)
    cin >> A[i].x >> A[i].y >> B[i].x >> B[i].y;


  for(int i=0; i<n; i++)
    for(int j=i+1; j<n; j++)
            adj[i][j] = adj[j][i] = intersect(A[i],B[i],A[j],B[j]);

  compSize=0;
  dfs(n-1);

  cout << n-compSize << endl;

  return 0;
}
```

Tango to Evora

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	128 MB	standardni ulaz	standardni izlaz

Problem D Interval Graph

Interval graf – uvod



Ideja 1: Booklet

Ideja 2: Prvo primetimo da listovi u našem drvetu nemaju velikog uticaja na samo drvo.

Listove možemo smatrati kao intervale koji sadrže samo jednu tacku.
Da bi došli do punog zakljucka, treba da uocimo odredjena pravila.
Da bi intervali imali zajednicke tacke, postoje cetiri moguca odnosa izmedu intervala.
Ili se nalaze jedan unutar drugog (2 slucaja), ili se delimicno preklapaju (2 slucaja).
Primetimo da za cvor koji nije list, tj. ima potomke, i nije koren, tj. ima pretka,
prva dva slucaja ne dolaze u obzir za bilo koji od odnosa sa cvorovima sa kojima je povezan, jer bi oba slucaja implicirala povezanost potomaka i pretka, što je kontradikcija. TROUGAO, tj. CIKLUS!!!

Moguc je specifican slucaj kada cvor obuhvata sve potomke i pretka.
Medjutim, u tom slucaju bi potomci morali da budu listovi, a predak koren,
te cemo ovaj slucaj lako pokriti uklanjanjem listova.

Dakle, za cvor koji nije ni list, ni koren, preostaju dva slucaja:
svaki od cvorova povezanih sa njim je interval koji ga delimicno preklapa ili levo ili desno.

Kako vec znamo da ce se preklapati sa jedne strane sa svojim pretkom,
ostaje samo još jedan cvor sa kojim ce on moci da ima zajednicke tacke.
Dakle, dobili smo uslov: svaki cvor koji nije koren i nije list,
mora da ima najviše jednog potomka koji nije list.

Što se samog korena tice, on ce morati da ima najviše
dva potomka koji nisu listovi,
i ukoliko ima jednog potomka, moci cemo da prenesemo svojstvo korena na njegovog potomka,
tj. njegov potomak ce moci da ima najviše dva potomka koji nisu listovi.
Ovo možemo vizuelizovati kao odnos u kome je koren interval
koji potpuno pripada intervalu koji je njegov potomak,
a potomci njegovog potomka su intervali koji delimicno preklapaju po jednu
od strana tog intervala.
Pošto su nam eksplicitno dati potomci svakog cvora,
ne moramo da radimo dfs kako bi pronašli listove,
pa cemo samo bfs-om proveriti da li svi preostali cvorovi ispunjavaju uslov

Složenost: O(n) za bfs

Fragile

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
2 s	1000 MB	standardni ulaz	standardni izlaz

Potrebno je izračunati na koliko načina se mogu postaviti n kraljica na šahovsku tablu dimenzije n*n, tako da se nikoje dve ne napadaju. Posmatrajmo sve moguće postavke kraljica i sortirajmo ih leksikografski, tj. sortiramo po pozicji prve kraljice, pa po poziciji druge kraljice, itd. Pozicije su uređeni parovi (red, kolona), gde se pozicije sortiraju prvo po manjem redu, pa po manjoj koloni. Polje (1,1) se nalazi gore-levo na šahovskoj tabli. Potrebno je ispisati k-tu konfiguraciju.

Ulaz

U prvoj i jedinoj liniji ulaza se učitavaju brojevi n i k.

Izlaz

U prvom redu izlaza je potrebno ispisati na koliko načina se mogu postaviti n kraljica na tablu dimenzija n*n. Nakon toga je potrebno ispisati konfiguraciju k-te postavke kraljica, gde se prazno polje na šahovskoj tabli obeležava karakterom '.', dok polje na kojoj je kraljica treba da sadrži karakter '*'.

Ograničenja

1<=n<=13

Ulaz

13 10000

Izlaz

73712

..*.....

.....*....

.....*.

.....*...

.*.....

...*.....

.....*.....

.....*.....

.....*

.....*..

*.....

....*.....

.....*.....

Napomena:
ULAZ

1 1
IZLAZ
1
*

za n=1, k>n je besmisleno i ocekivati takav test primer, ali posto ne postoji k-ta postavka ispisati praznu nisku
ULAZ
1 100
IZLAZ
1

Slicno za n=2, 3
ULAZ
3 2
IZLAZ
0

Ideja 1:
Brute force strategija:
Kako bi postavili n kraljica na tablu dimenzija n x n, u svakom redu i koloni mora biti tacno jedna kraljica.
Na svakoj levoj i desnoj dijagonali, tj. odsečcima dijagonala takođe mora biti tacno jedna kraljica.
Krenucemo od prvog reda i prve pozicije u prvom redu i rekurzivno generisati svako moguce rešenje koje zadovoljava uslove da se nijedna kraljica ne napada.

Pošto se rekurzija vrši po redovima, nema potrebe da čuvamo uslove jedinstvenosti kraljice u redu, već samo po levoj i desnoj dijagonali, i kolonama. U nizu (npr. Queens) čuvace se poredak kraljica po redovima, i njega cemo kopirati u niz kth kada dodemo do k-tog rešenja koje se traži.

U nizu forbid1 cuvace se uslov jedinstvenosti kraljice u svakoj koloni (uslov postojanja je nepotreban jer ce se u svakoj koloni svakako generisati kraljica zbog drugih uslova), ovaj niz je lako konstruisati i samo treba da upišemo 1 u kolonu kraljice koju smo trenutno izabrali.
Uvedimo nizove forbid2 i forbid3 koji će čuvati uslove, tj. zabrane izbora kraljica koje bi po dijagonalama napadale prethodne kraljice. Dovoljno je zapamtiti pocetni indeks dijagonale koju biramo (taj pocetni indeks može biti i van granica table, pa cemo zato ova dva niza proširiti sa leve i desne strane za po n). Onda cemo lako moci da proverimo da li neki indeks u trenutnom redu pripada zabranjenoj dijagonali.

Složenost bi u teorijskom smislu bila eksponencijalna, tj. n^n, jer imamo n redova iz kojih biramo n kraljica. Međutim, zbog uslova zabrane kolone, svaki red ce imati makar jedan manje izbor, cime se složenost smanjuje na faktorijalnu. Ako ukljucimo i zabranu po dijagonalama, dobijamo još manju kompleksnost.
U strogo prakticnom smislu, vidimo da ce složenost moci da se opiše izrazom
n * (k1 + (n - k1) * (k2 + (n - k2) * ...), gde je ki broj mogucih izbora za i-ti red (koji naravno nije konstantan, ali može se aproksimovati).
Dalje, možemo da odredimo broj operacija koje su potrebne za svako n, jer postoji samo 13 mogucnosti, a složenost ne zavisi od k. Ako kao operaciju definišemo svaku iteraciju petlje koja pokušava da postavi kraljicu na i-tu kolonu u it-tom redu, dobijamo:
1: 1
2: 6
3: 18
4: 60
5: 220
6: 894
7: 3584
8: 15720
9: 72378
10: 348150
11: 1806706
12: 10103868
13: 59815314

Hymn for the weekend

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
0,6 s	64 MB	standardni ulaz	standardni izlaz

Takmičari su dobili `N` kartica, obeleženih brojevima od 1 do `N` koje moraju obojiti do kraja takmičarskog dana.
Poznato je da među njima postoji `M` parova karti koje moraju biti obojene istom bojom. Napisati program koji pronalazi najveći broj broja manjih od `N` koje se mogu upotrebiti za bojenje svih karti.

Ulaz

U prvom redu standardnog ulaza dati su celi brojevi `N` i `M` (). U narednih `M` redova su data po dva cela broja – obeležja dve karte, koje moraju da budu iste boje. Među datim parovima, neki parovi se mogu ponoviti.

Izlaz

Na standarni izlaz ispisati jedan ceo broj, najveći broj boja koje se mogu upotrebiti za bojenje svih karti.

Primer

Ulaz

```
12 5
2 1
2 3
1 2
1 3
1 4
```

Izlaz

```
9
```

Karte 1,2,3,4 se boje istom bojom. Zato je najveći ukupan broj boja 9.

Ideja:

Pravimo graf (čvorovi bili karte, a grane bi postojale između dva čvora ukoliko oni treba da se oboje istom bojom).
Potrebno je da izračunamo broj komponenata povezanosti grafa čiji bi čvorovi bili karte, a grane bi postojale između dva čvora ukoliko oni treba da se oboje istom bojom.

Međutim, eksplicitno predstavljanje grafa je nemoguće zbog potencijalno velikog broja čvorova (10^9), pa se može koristiti drugi pristup. Treba da cuvamo samo cvorove sa granama, a ostale tretiramo zasebno.

Koristicemo mapu da bi cuvali cvorove i odredicemo broj komponenti povezanosti obicnim dfs-om. Za svaki cvor (koji sadrži granu) pokrenucemo dfs, ako je on neposecen, onda on pripada jednoj komponenti povezanosti, i dfs-om posetimo sve cvorove te komponente. Ako je posecen, znaci da smo vec uracunali komponentu povezanosti kojoj taj cvor pripada, i njega cemo preskociti.

Složenost: O(mlogm) za obradu ulaza, jer koristimo mapu, iza koje je balansirano bst. O(m + |V|) za dfs, gde je |V| broj cvorova iz kojih polaze grane.

Knez Igor

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
60 s	1000 MB	standardni ulaz	standardni izlaz

Dat je lavirint širine A i visine B. Odrediti da li postoji izlaz iz lavirinta počevši od datih koordinata X i Y. Koordinate 0,0 predstavljaju gornje levo polje lavirinta.

Ulaz

U prvom redu standardnog ulaza nalaze se vrednosti A i B odvojene razmakom.
U drugom redu standardnog ulaza nalaze se koordinate X i Y odvojene razmakom.
Topologija lavirinta je data u sledećih B redova koji sadrže po A karaktera od kojih svaki predstavlja po jedno polje lavirinta.
Karakter 'X' predstavlja zid, a razmak (' ') predstavlja prolaz.

Izlaz

U prvom redu standardnog izlaza treba da se nalazi reč "DA" ako postoji izlaz počevši od polja (X, Y) ili reč "NE" ako izlaz ne postoji.

Ograničenja

1 <= A,B <= 256
0 <= X < A
0 <= Y < B

ex

Ulaz

```
10 5
1 1
 XXX  XXX
```

X XX X
XXX X X X
X   XX
XXXXXXX XX

Izlaz
DA

Ideja 1: Backtracking (vremensko ograničenje 60 s)

Ideja 2:
BFS
Idemo po slobodnim poljima dokle možemo. Ako u jednom trenutku nemamo više neposećenih polja koji su sledeci po bfs udaljenosti, onda nema rešenja.
U suprotnom, ako dođemo do praznog polja na obodu lavirinta, onda možemo napustiti lavirint. možemo direktno proveravati da li je grana na obodu, ili možemo ograditi lavirint karakterima Y, i zaustaviti algoritam kad posetimo polje Y, koje je van lavirinta.
Složenost: O(a * b)

Je veux

Vremensko ograničenje	Memorijsko ograničenje	ulaz	izlaz
1 s	1000 MB	standardni ulaz	standardni izlaz

Popularna TV emisija Nindža ratnici je kako bi podigla gledanost uvela novu igru u šou: Lavirint. U toj igri takmičari treba da što pre nađu izlaz iz lavirinta. Lavirint je pravougaonog oblika, i izdeljen u mrežu od N x M jednakih soba (N redova po M soba). Svaka soba je povezana vratima sa susednim sobama. Dakle, sobe u sredini lavirinta su povezane sa četiri susedne sobe, sobe na ivicama sa tri, a one u ćošku sa dve susedne sobe. Na sredini svake sobe se nalazi sto na kome je jedan ili više kimono pojaseva. Takmičari kreću od ulaza u lavirint koji ih uvodi u gornju levu sobu lavirinta, a izlaz iz lavirinta je uvek u krajnje desnoj donjoj sobi lavirinta. Pobednik je takmičar koji u najkraćem vremenu stigne do cilja, a ukoliko više takmičara stigne u isto vreme pobednik je onaj koji usput sakupi najviše kimono pojaseva.

Potrebno je pronaći optimalnu putanju kojom takmičar treba da se kreće kako bi pobedio u ovoj igri i sebi osigurao laskavu titulu nindža ratnika.

Napomena: Smatrati da je vreme koje je takmičaru potrebno da prođe kroz sobu i pokupi sve pojaseve u njoj konstantno, bez obzira na koja vrata takmičar ulazi, i na koja izlazi. Takođe smatra se da su takmičari sposobni da ponesu proizvoljan broj kimono pojaseva sa sobom.

Ulaz

U prvom redu su zadata dva prirodna broja N i M.

U narednih N redova je u svakom zadato po M prirodnih brojeva koji predstavljaju broj kimono pojaseva u svakoj od soba u lavirinu.

Izlaz

Potrebno je ispisati instrukcije kako takmičar treba da se kreće od početne sobe do sobe u kojoj je izlaz iz lavirinta. Na izlazu treba ispisati po jednu od komandi: Gore, Dole, Levo, Desno u svakom redu.

Ukoliko postoji više optimalnih putanja potrebno je pronaći i ispisati samo jednu od njih.

Ograničenja

N, M < 1 000

Ulaz

```
4 5
1 1 3 1 2
2 5 3 1 1
6 1 2 4 2
1 1 1 3 2
```

Izlaz

```
Dole
Desno
Desno
Dole
Desno
Dole
Desno
```

Ideja 1:
Zadatak na prvi pogled izgleda kao da se napada Dijkstra algoritmom.
Ali, s obzirom da je prvi uslov po kome se rangiraju putevi kroz matricu dužina puta, pokrenucemo klasican bfs sa pamcenjem najveceg moguceg broja pojaseva koji mogu da se sakupe do te sobe.
Poseticemo i vec posecene cvorove (koji se obraduju u tom koraku), jer je moguce da smo pronašli veci broj pojaseva od vec postavljenog.
Bitno je i pamtiti čvor iz kojeg smo došli do maksimalnog broja pojaseva u nekoj sobi.
jer treba da nađemo put, a ne broj pojaseva. Zato je najlakše krenuti od kraja, tj. poslednje sobe u bfs-u.
Onda nećemo morati da pravimo dodatne nizove i radimo obrtanje smera pri procesu štampanja.
Složenost: O(nm)

Ideja 2: dinamičko programiranje